Project Title

Optimizing Multilayer Perceptrons for Class Posterior Estimation and Minimizing Classification Error: A Comparative Study with Theoretical Classifiers

Project by
Shiva Kumar Dande

Project Completion Date:

March 2023

Problem Statement

In this exercise, you will train many multilayer perceptrons (MLP) to approximate the class label posteriors, using maximum likelihood parameter estimation (equivalently, with minimum average cross-entropy loss) to train the MLP. Then, you will use the trained models to approximate a MAP classification rule in an attempt to achieve minimum probability of error (i.e. to minimize expected loss with 0-1 loss assignments to correct-incorrect decisions).

Data Distribution: For $C = 4$ classes with uniform priors, specify Gaussian class-conditional pdfs for a 3-dimensional real-valued random vector $\mathbf{x}$ (pick your own mean vectors and covariance matrices for each class). Try to adjust the parameters of the data distribution so that the MAP classifier that uses the true data pdf achieves between $10\% - 20\%$ probability of error.

MLP Structure: Use a 2-layer MLP (one hidden layer of perceptrons) that has P perceptrons in the first (hidden) layer with smooth-ramp style activation functions (e.g., ISRU, Smooth-ReLU, ELU, etc). At the second/output layer use a softmax function to ensure all outputs are positive and add up to 1. The best number of perceptrons for your custom problem will be selected using cross-validation.

Generate Data: Using your specified data distribution, generate multiple datasets: Training datasets with $100, 200, 500, 1000, 2000, 5000$ samples and a test dataset with 100000 samples. You will use the test dataset only for performance evaluation.

Theoretically Optimal Classifier: Using the knowledge of your true data pdf, construct the minimum-probability-of-error classification rule, apply it on the test dataset, and empirically estimate the probability of error for this theoretically optimal classifier. This provides the aspirational performance level for the MLP classfier.

Model Order Selection: For each of the training sets with different number of samples, perform 10-fold cross-validation, using minimum classification error probability as the objective function, to select the best number of perceptrons (that is justified by available training data).

Model Training: For each training set, having identified the best number of perceptrons using cross-validation, using maximum likelihood parameter estimation (minimum cross-entropy loss) train an MLP using each training set with as many perceptrons as you have identified as optimal for that training set. These are your final trained MLP models for class posteriors (possibly each with different number of perceptrons and different weights). Make sure to mitigate the chances of getting stuck at a local optimum by randomly reinitializing each MLP training routine multiple times and getting the highest training-data log-likelihood solution you encounter.

Performance Assessment: Using each trained MLP as a model for class posteriors, and using the MAP decision rule (aiming to minimize the probability of error) classify the samples in the test set and for each trained MLP empirically estimate the probability of error.

Report Process and Results: Describe your process of developing the solution; numerically and visually report the test set empirical probability of error estimates for the theoretically optimal and multiple trained MLP classifiers. For instance show a plot of the empirically estimated test P(error) for each trained MLP versus number of training samples used in optimizing it (with semilog-x axis), as well as a horizontal line that runs across the plot indicating the empirically estimated test P(error) for the theoretically optimal classifier.

Note: You may use software packages for all aspects of your implementation. Make sure you use tools correctly. Explain in your report how you ensured the software tools do exactly what you need them to do.

Train and Test / Validation Data Generation:
The datasets are made up of 3D real-valued vectors x that belong to one of four distinct classes with equal priors and were created using a Gaussian Mixture Model. These datasets are produced by the A3 1 Generate data.py script found in the Appendix section.

Several datasets containing 100, 200, 500, 1,000, 2,000, and 5,000 samples are generated for training purposes. To evaluate the final performance, a 100,000 sample test dataset is produced.

Neural Network Architecture
The class posteriors are estimated via a two-layer neural network. Four neurons make up the output layer, which is followed by a SoftMax layer that determines probability for the four class labels. The ELU (Exponential Linear Unit) activation function is used in the first layer to introduce non-linearity.

The ELU activation function was chosen because it is uniformly smooth and differentiable. It also has a wide range, which lessens the issue of vanishing gradients as the gradient gets close to a value of 0.

Cross-validation and Model Order Selection
Each dataset's hyperparameter, the first layer's perceptron count (the hidden layer between input and output), is determined using 10-fold cross validation. The training dataset is divided into 10 equal pieces using this procedure, one of which serves as the validation set and the other nine as the training set. To determine the smallest cross-entropy loss for each iteration, model parameters are generated using the training set and tested on the validation set. 10 iterations of training the model with each fold. The average cross-entropy loss is determined to assess the model's performance after training the model 10 times with each fold serving as the validation set once. To determine the best number for each situation, the number of neurons vary from 1 to 9. The model order that results in the lowest average test loss is identified using bar graphs.

software bundles
To provide built-in classes for creating the neural network and procedures for estimating learning parameters, PyTorch is used. Optim is a PyTorch function. Stochastic Gradient Descent (SGD) uses weight optimization to reduce loss. As it modifies parameters for each sample, SGD introduces randomness, allowing the path to stray and aiding in escaping local minima. To speed up convergence, momentum is also provided to accelerate gradients in the desired direction. The Xavier Initialization procedure, which is used to uniformly choose random initial weight vectors at the beginning of a new training routine, is carried out by the init.xavier uniform_() method.

Theoretically Optimal Classifier

In order to compare the approximated MLP models for class posteriors, an ideal Bayesian classifier is trained using the following rule –

$R(D = 0|x)$

$R(D = 1|x)$ = A $*$ Transpose([$P(L = 0|x)$ $P(L = 1|x)$ $P(L = 2|x)$ $P(L = 3|x)$])

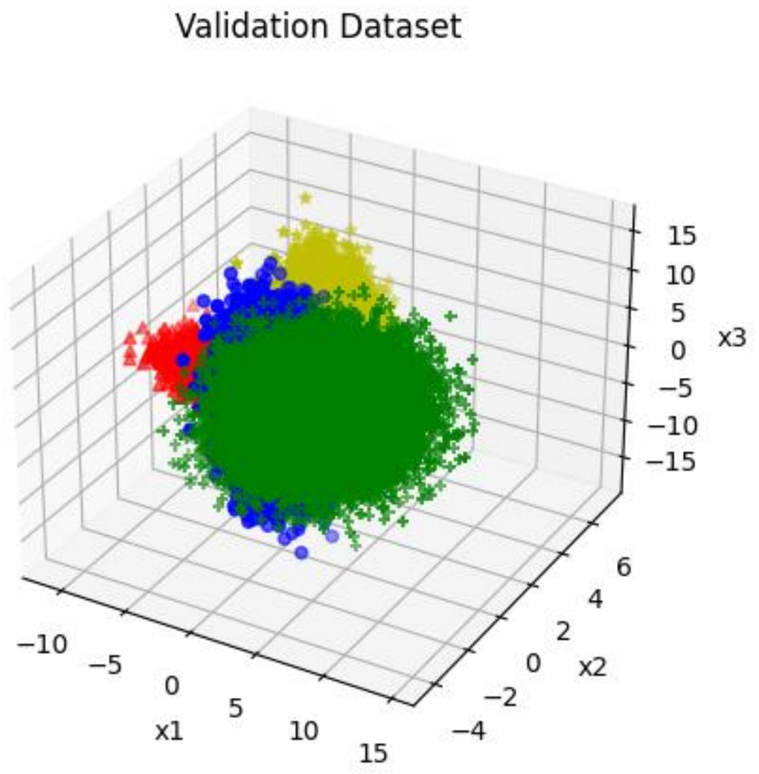
$R(D = 2|x)$

$R(D = 3|x)$

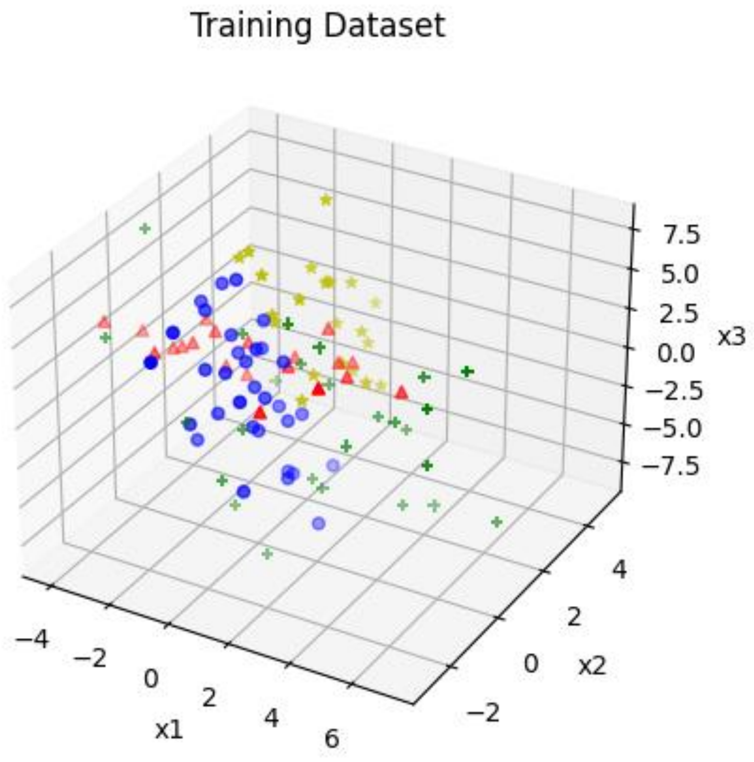
Risk = Loss Matrix * Class Posteriors
(0-1 Loss)

Decision(x) = $\underset{d\ \varepsilon\ \{0,1,2,3\}}{\text{argmin}}$ Risk (Decision(x) = d | x)

Validation Dataset: 10000 samples



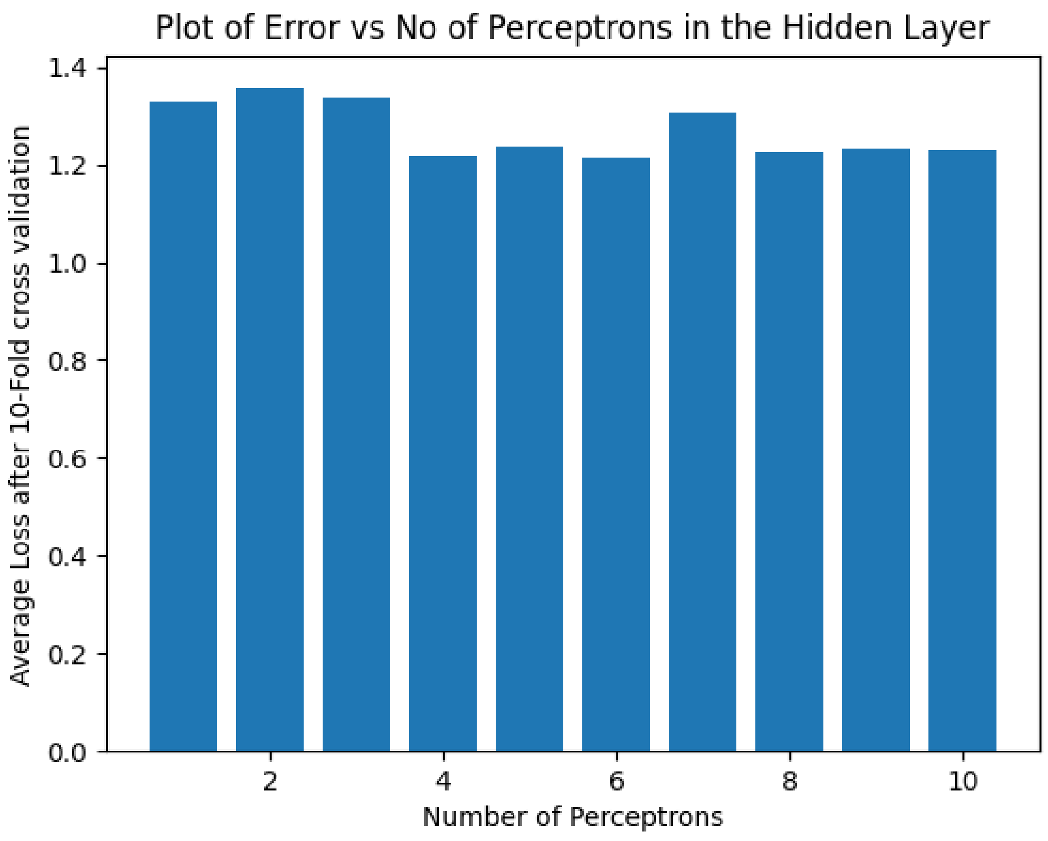
No of samples in Training Data: 100



Output:

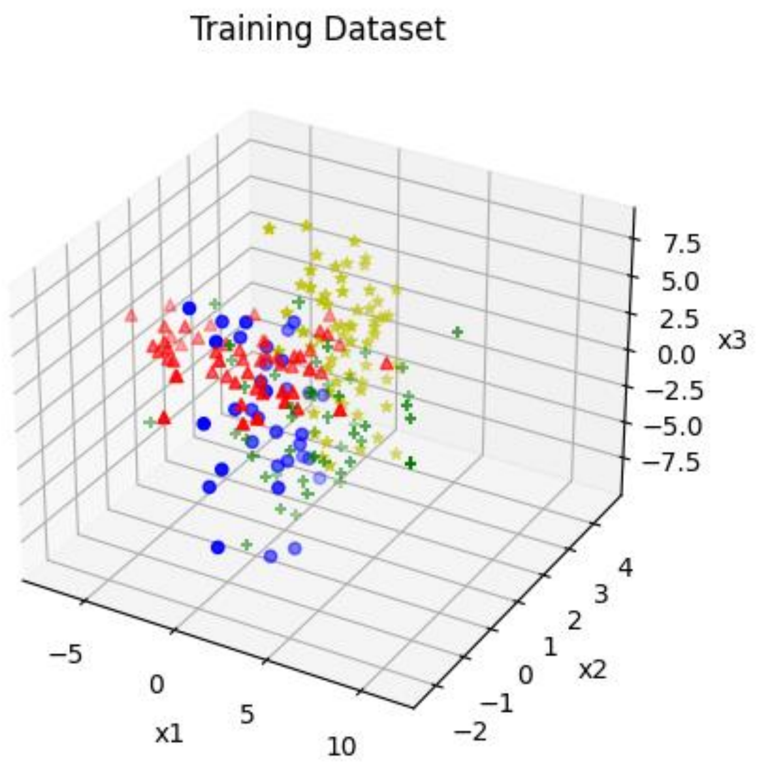
```
--------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 1
--------------------------------------------------------------------
Mean error for 1 perceptrons: 1.3287
--------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 2
--------------------------------------------------------------------
Mean error for 2 perceptrons: 1.3556
--------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 3
--------------------------------------------------------------------
Mean error for 3 perceptrons: 1.3382
--------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 4
--------------------------------------------------------------------
Mean error for 4 perceptrons: 1.2164
--------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 5
--------------------------------------------------------------------
Mean error for 5 perceptrons: 1.2386
--------------------------------------------------------------------
```

```
Number of Neurons / Perceptrons in the Hidden Layer : 6
------------------------------------------------------------------
Mean error for 6 perceptrons: 1.2149
------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 7
------------------------------------------------------------------
Mean error for 7 perceptrons: 1.3073
------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 8
------------------------------------------------------------------
Mean error for 8 perceptrons: 1.2251
------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 9
------------------------------------------------------------------
Mean error for 9 perceptrons: 1.2327
------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 10
------------------------------------------------------------------
Mean error for 10 perceptrons: 1.2277
Optimal number of Perceptrons = 6: VALIDATION LOSS = 1.2149, VALIDATION ACCURACY = 0.5600
```



```
FINAL LOSS IS 0.99788498878479, ACCURACY IS 0.74209
Confusion Matrix:
[[15233  7680  1827   364]
 [  301 21371   275  2782]
 [  906   639 23292    14]
 [ 5580  4636   787 14313]]
Error:
0.25791 (25.79 %)
```

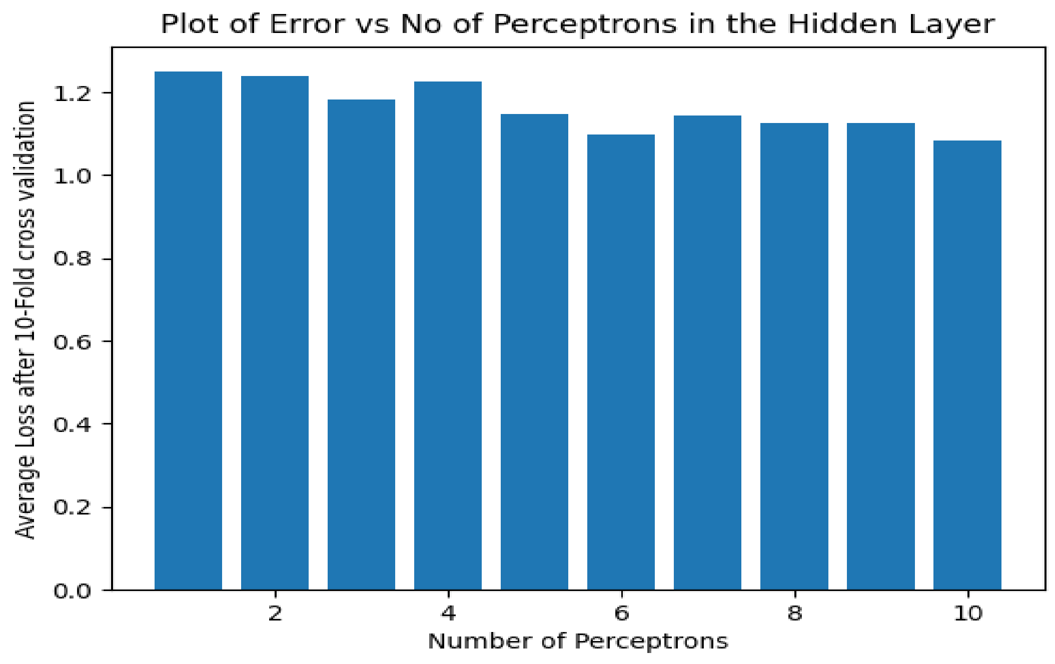
No of samples in Training Data: 200



```
---------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 1
---------------------------------------------------------------------
Mean error for 1 perceptrons: 1.2491
---------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 2
---------------------------------------------------------------------
Mean error for 2 perceptrons: 1.2397
---------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 3
---------------------------------------------------------------------
Mean error for 3 perceptrons: 1.1819
---------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 4
```

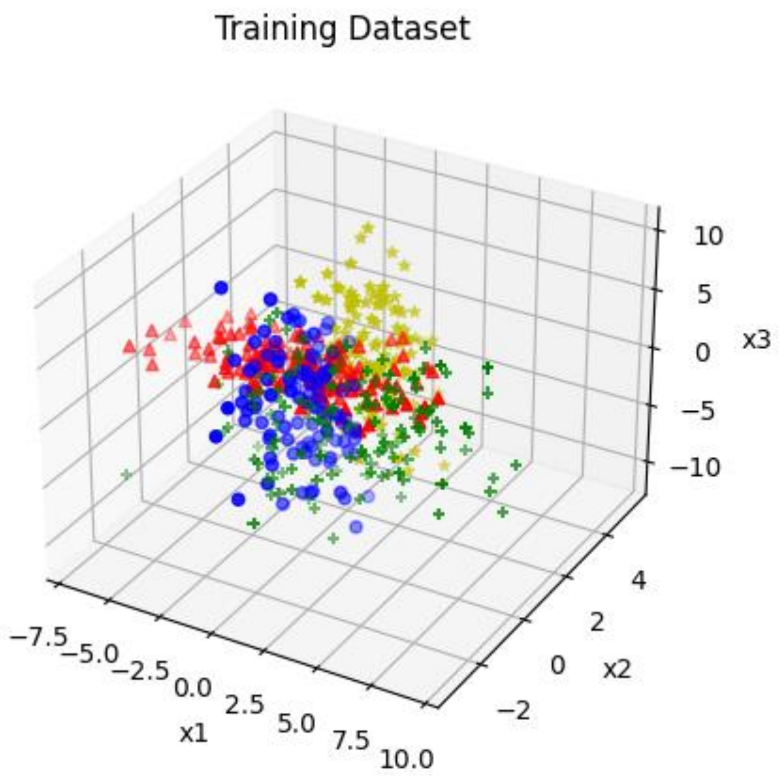
```
--------------------------------------------------------------------
Mean error for 4 perceptrons: 1.2259
--------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 5
--------------------------------------------------------------------
Mean error for 5 perceptrons: 1.1469
--------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 6
--------------------------------------------------------------------
Mean error for 6 perceptrons: 1.0972
--------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 7
--------------------------------------------------------------------
Mean error for 7 perceptrons: 1.1417
--------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 8
--------------------------------------------------------------------
Mean error for 8 perceptrons: 1.1273
--------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 9
--------------------------------------------------------------------
Mean error for 9 perceptrons: 1.1244
--------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 10
--------------------------------------------------------------------
Mean error for 10 perceptrons: 1.0847
```

Optimal number of Perceptrons = 10 : VALIDATION LOSS = 1.0847, VALIDATION ACCURACY = 0.7050



FINAL LOSS IS 0.9949209094047546, ACCURACY IS 0.74505
Confusion Matrix:
```
[[14316  7485  1899  1404]
 [  565 20574   171  3419]
 [  848  1120 22812    71]
 [ 3349  4535   629 16803]]
```
Error:
0.25495 (25.49 %)

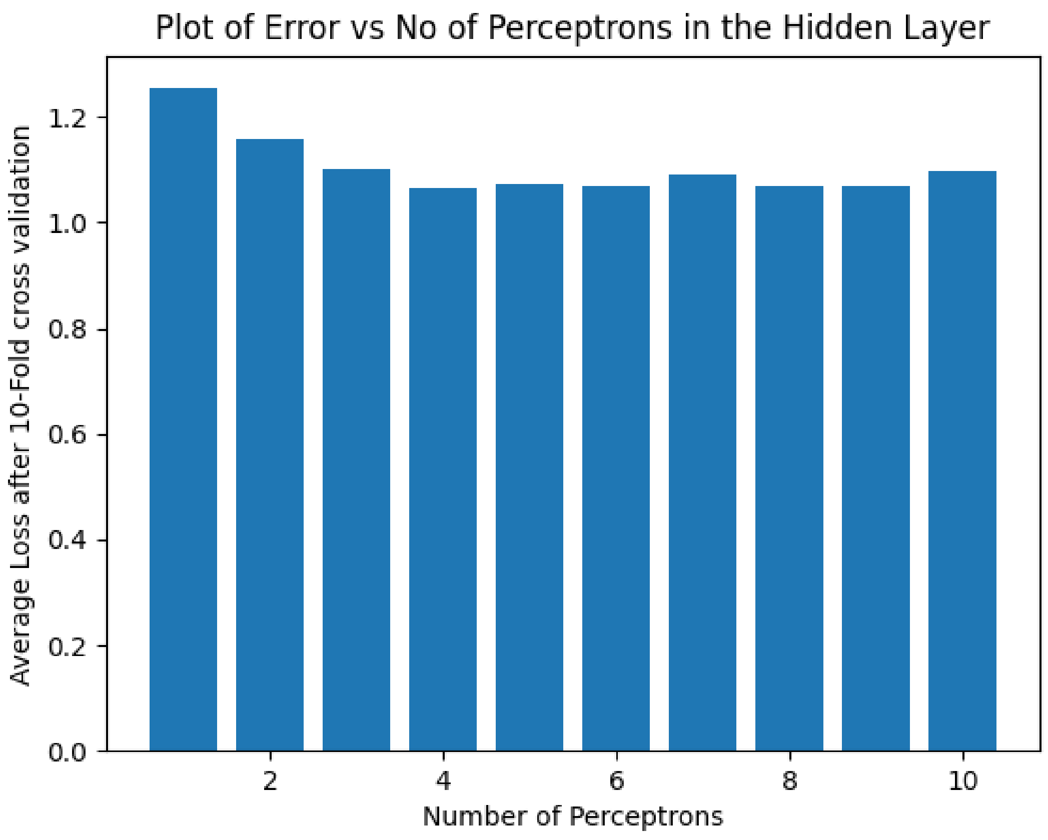
No of samples in Training Data: 500



Output:

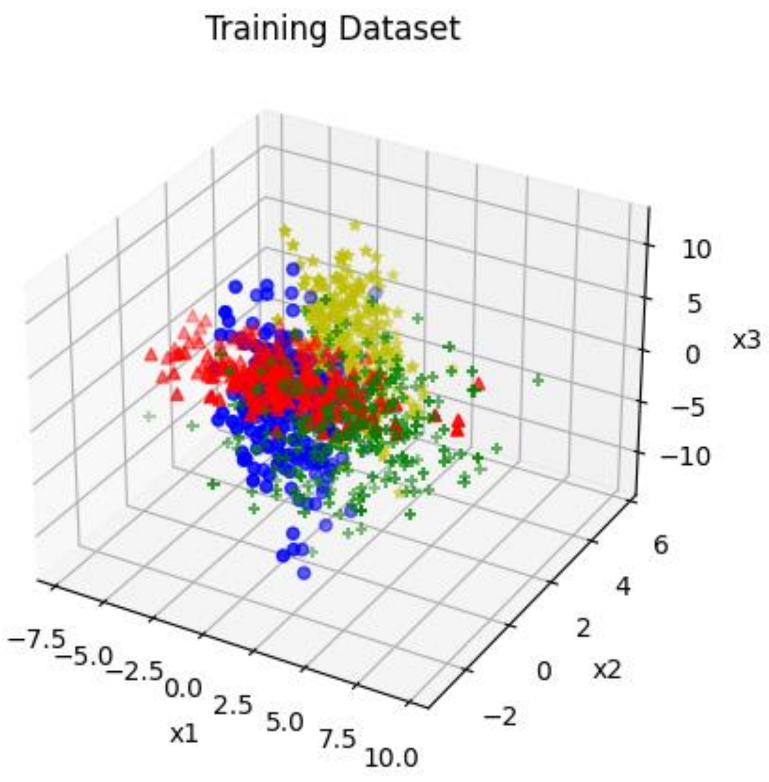
```
--------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 1
--------------------------------------------------------------------
Mean error for 1 perceptrons: 1.2534
--------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 2
--------------------------------------------------------------------
Mean error for 2 perceptrons: 1.1593
--------------------------------------------------------------------
```

```
Number of Neurons / Perceptrons in the Hidden Layer : 3
--------------------------------------------------------------------
Mean error for 3 perceptrons: 1.1016
--------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 4
--------------------------------------------------------------------
Mean error for 4 perceptrons: 1.0668
--------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 5
--------------------------------------------------------------------
Mean error for 5 perceptrons: 1.0728
--------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 6
--------------------------------------------------------------------
Mean error for 6 perceptrons: 1.0679
--------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 7
--------------------------------------------------------------------
Mean error for 7 perceptrons: 1.0900
--------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 8
--------------------------------------------------------------------
Mean error for 8 perceptrons: 1.0679
--------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 9
--------------------------------------------------------------------
Mean error for 9 perceptrons: 1.0696
--------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 10
--------------------------------------------------------------------
Mean error for 10 perceptrons: 1.0966
Optimal number of Perceptrons = 4 : VALIDATION LOSS = 1.0668, VALIDATION
ACCURACY = 0.6880
```



```
FINAL LOSS IS 0.9724533557891846, ACCURACY IS 0.76633
Confusion Matrix:
[[16881  6140  1779   304]
 [  645 23061   195   828]
 [ 1401   330 23106    14]
 [ 4935  6203   593 13585]]
Error:
0.23367000000000004 (23.36 %)
```

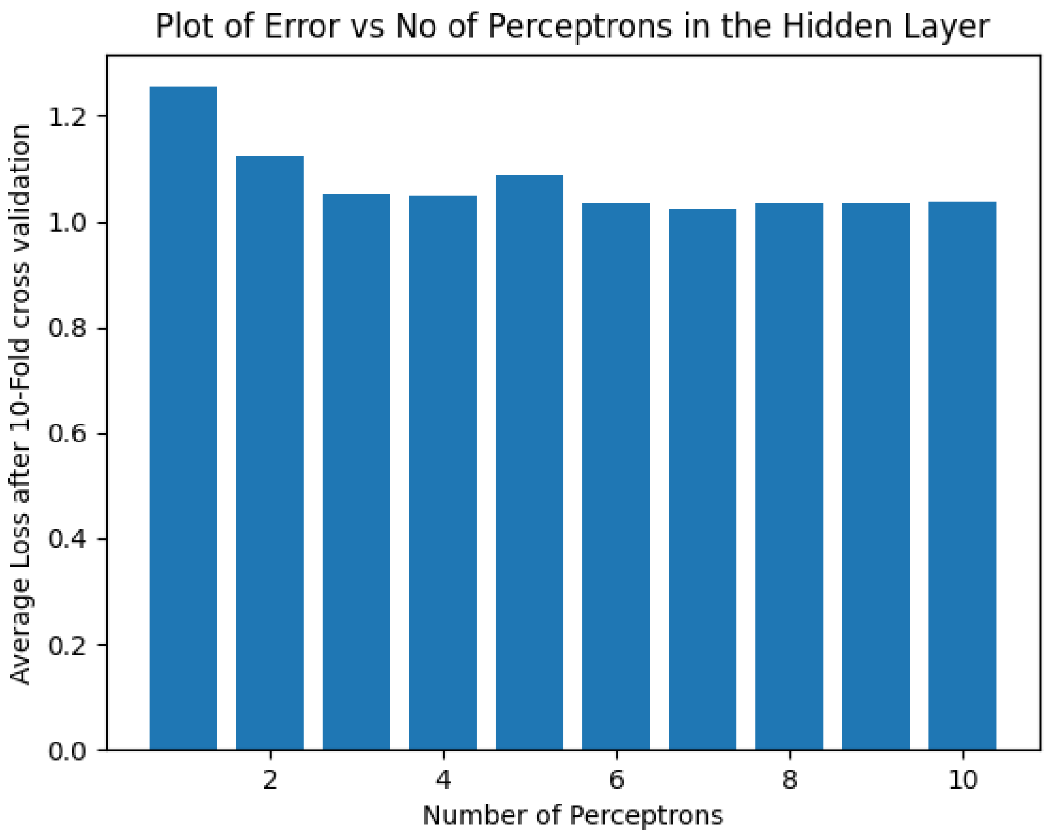
No of samples in Training Data: 1000



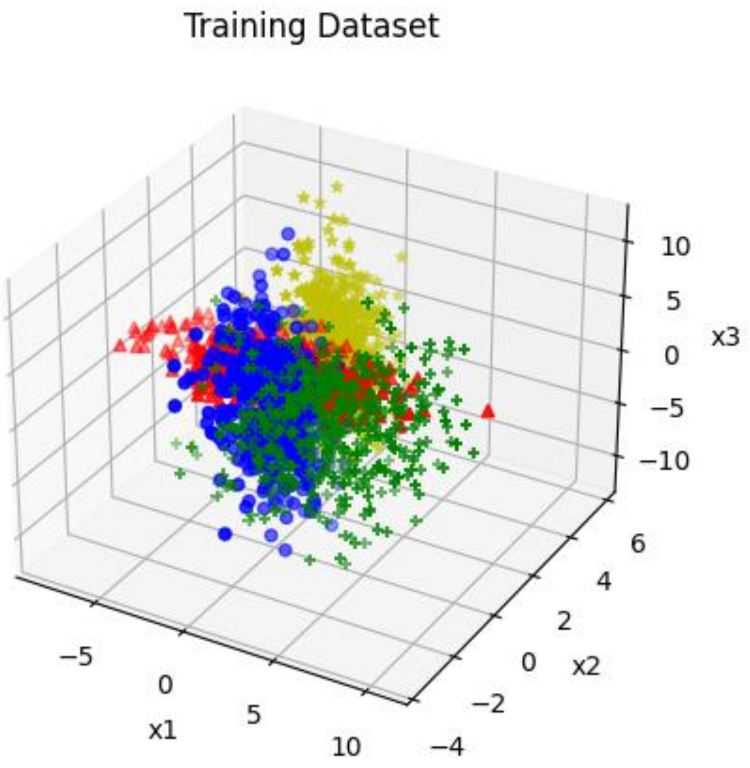
```
-------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 1
-------------------------------------------------------------------
Mean error for 1 perceptrons: 1.2540
-------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 2
-------------------------------------------------------------------
Mean error for 2 perceptrons: 1.1236
-------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 3
-------------------------------------------------------------------
Mean error for 3 perceptrons: 1.0523
-------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 4
-------------------------------------------------------------------
Mean error for 4 perceptrons: 1.0495
-------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 5
-------------------------------------------------------------------
Mean error for 5 perceptrons: 1.0883
-------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 6
-------------------------------------------------------------------
```

```
Mean error for 6 perceptrons: 1.0327
-------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 7
-------------------------------------------------------------------
Mean error for 7 perceptrons: 1.0248
-------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 8
-------------------------------------------------------------------
Mean error for 8 perceptrons: 1.0355
-------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 9
-------------------------------------------------------------------
Mean error for 9 perceptrons: 1.0335
-------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 10
-------------------------------------------------------------------
Mean error for 10 perceptrons: 1.0384
```

Optimal number of Perceptrons = 7 : VALIDATION LOSS = 1.0248, VALIDATION ACCURACY = 0.7390



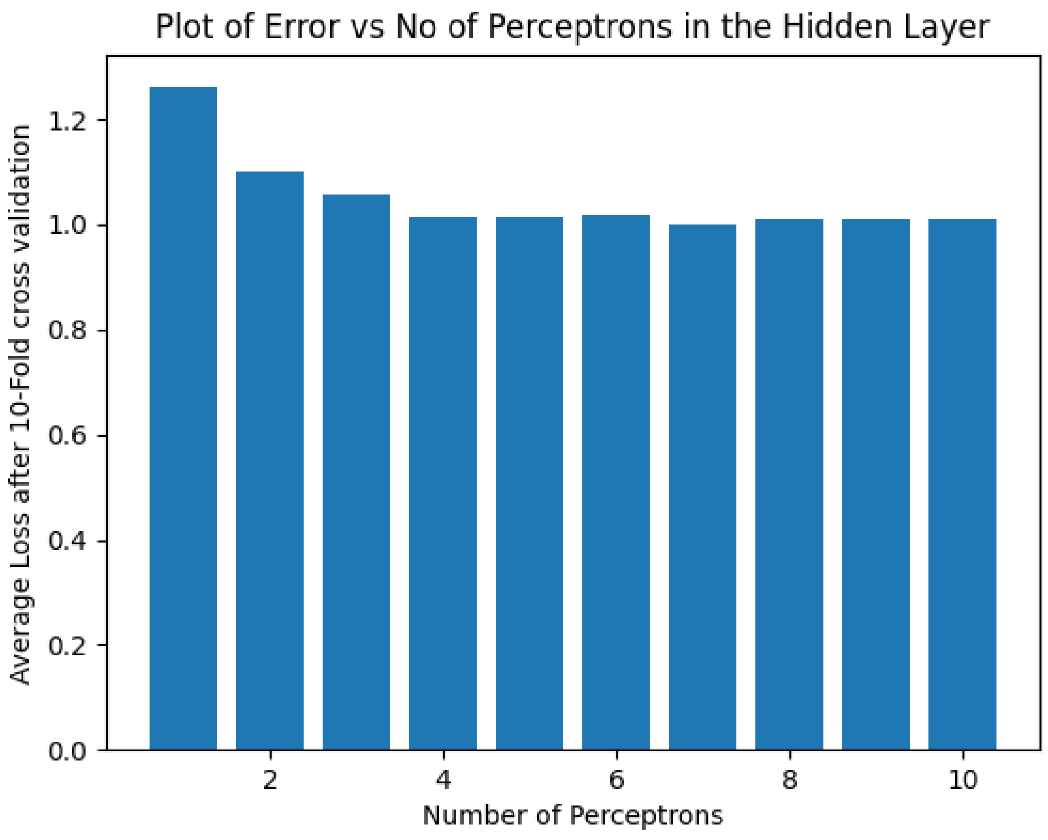
```
FINAL LOSS IS 0.9706096649169922, ACCURACY IS 0.7683
Confusion Matrix:
[[17308  5358  1838   600]
 [  383 23615   315   416]
 [ 1647   443 22744    17]
 [ 5795  5609   749 13163]]
Error:
0.23170000000000002 (23.17 %)

No of samples in Training Data: 2000
```



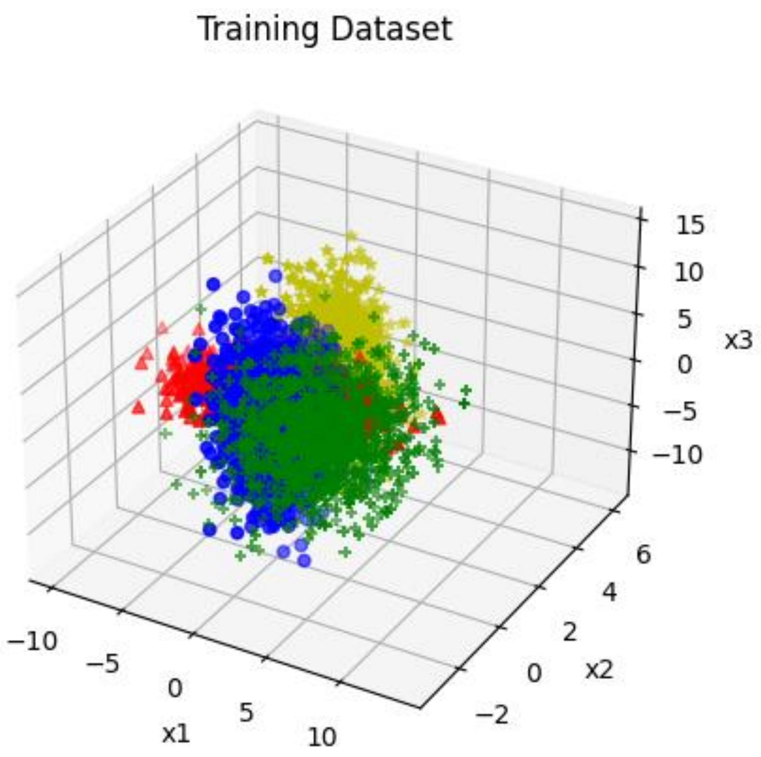
```
--------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 1
--------------------------------------------------------------------
Mean error for 1 perceptrons: 1.2607
--------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 2
--------------------------------------------------------------------
Mean error for 2 perceptrons: 1.0997
--------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 3
--------------------------------------------------------------------
Mean error for 3 perceptrons: 1.0562
--------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 4
--------------------------------------------------------------------
Mean error for 4 perceptrons: 1.0141
--------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 5
--------------------------------------------------------------------
```

```
Mean error for 5 perceptrons: 1.0131
--------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 6
--------------------------------------------------------------------
Mean error for 6 perceptrons: 1.0181
--------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 7
--------------------------------------------------------------------
Mean error for 7 perceptrons: 1.0006
--------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 8
--------------------------------------------------------------------
Mean error for 8 perceptrons: 1.0108
--------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 9
--------------------------------------------------------------------
Mean error for 9 perceptrons: 1.0120
--------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 10
--------------------------------------------------------------------
Mean error for 10 perceptrons: 1.0116
```

Optimal number of Perceptrons = 7: VALIDATION LOSS = 1.0006, VALIDATION ACCURACY = 0.7475



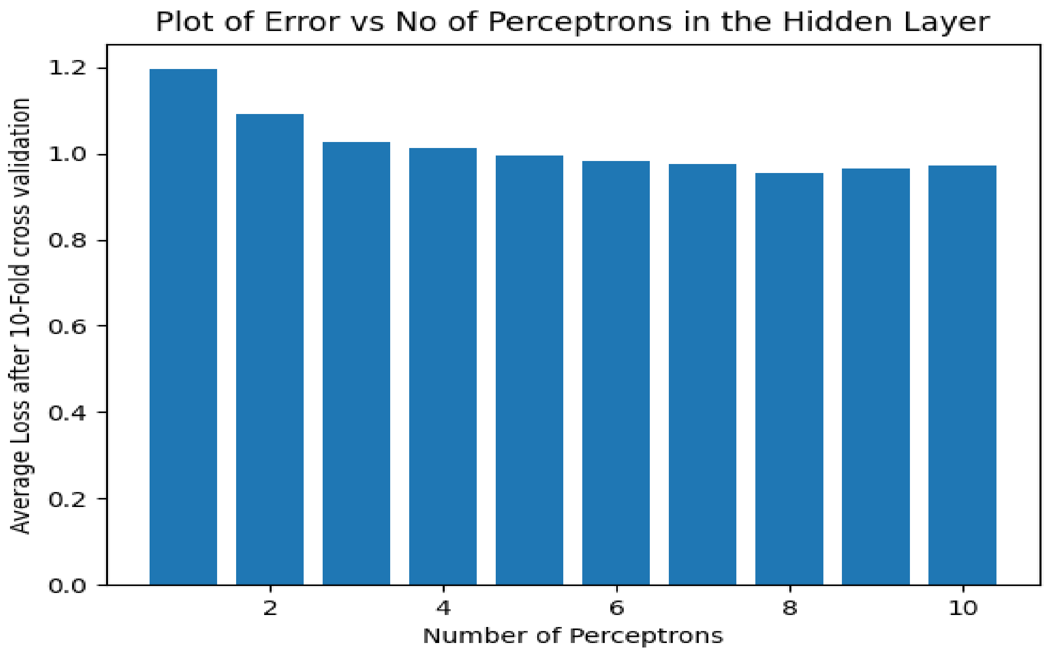
```
FINAL LOSS IS 0.9590023159980774, ACCURACY IS 0.78283
Confusion Matrix:
[[17347  4812  1933  1012]
 [  510 21603   220  2396]
 [ 1192   347 23269    43]
 [ 5416  3122   714 16064]]
Error:
0.21716999999999997 (21.71 %)

No of Samples in Training Data: 5000
```



```
----------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 1
----------------------------------------------------------------------
Mean error for 1 perceptrons: 1.1950
----------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 2
----------------------------------------------------------------------
Mean error for 2 perceptrons: 1.0912
----------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 3
----------------------------------------------------------------------
Mean error for 3 perceptrons: 1.0270
----------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 4
```

```
--------------------------------------------------------------------
Mean error for 4 perceptrons: 1.0115
--------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 5
--------------------------------------------------------------------
Mean error for 5 perceptrons: 0.9944
--------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 6
--------------------------------------------------------------------
Mean error for 6 perceptrons: 0.9811
--------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 7
--------------------------------------------------------------------
Mean error for 7 perceptrons: 0.9746
--------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 8
--------------------------------------------------------------------
Mean error for 8 perceptrons: 0.9555
--------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 9
--------------------------------------------------------------------
Mean error for 9 perceptrons: 0.9654
--------------------------------------------------------------------
Number of Neurons / Perceptrons in the Hidden Layer : 10
--------------------------------------------------------------------
Mean error for 10 perceptrons: 0.9705
```

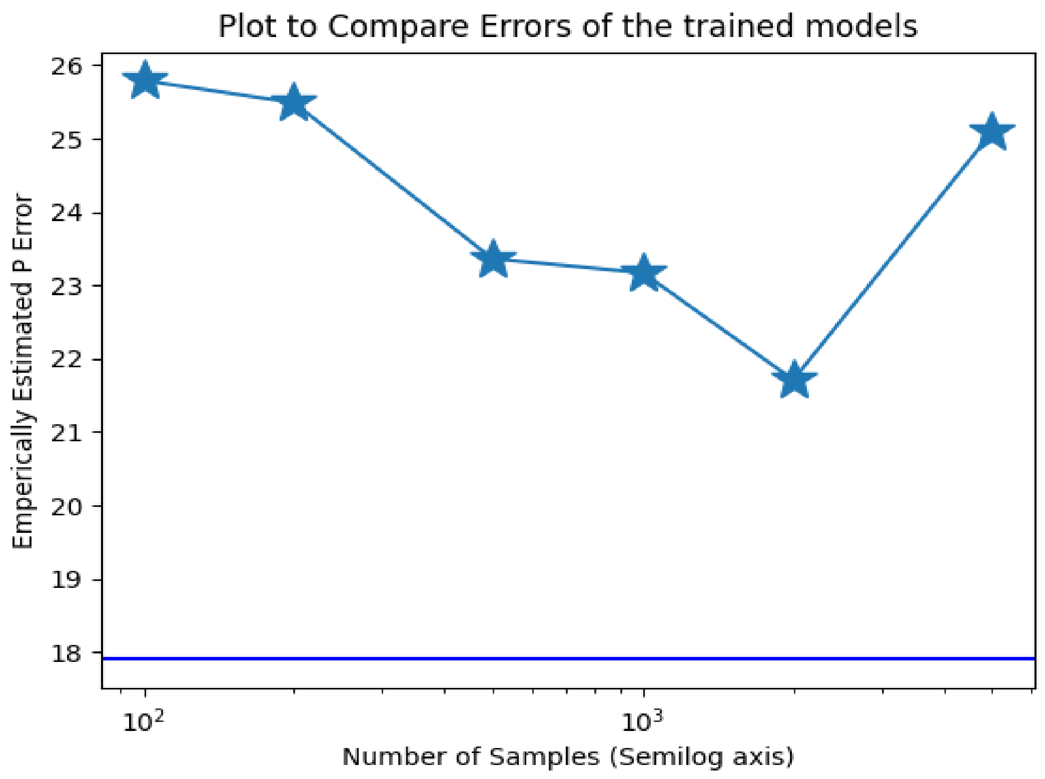
Optimal number of Perceptrons = 8 : VALIDATION LOSS = 0.9555, VALIDATION ACCURACY = 0.7920



```
FINAL LOSS IS 0.9956095218658447, ACCURACY IS 0.74491
Confusion Matrix:
[[14355  7102  2034  1613]
 [  655 23284   308   482]
 [  749   876 23111   115]
 [ 3274  7672   629 13741]]
Error:
0.25509000000000004 (25.09 %)

Output from Theoretical Classifier:
Average Expected Risk: 0.17903905515804705

Confusion Matrix:
[[0.7790392  0.05434914 0.05959519 0.16637699]
 [0.12531867 0.88733875 0.00776629 0.12549376]
 [0.0625     0.00505479 0.92805119 0.01595829]
 [0.03314213 0.05325731 0.00458734 0.69217096]]
```

Comparison of Theoretical Classifier and MLP Models

The blue data points in the image show the experimentally calculated loss with a semi-log x-axis to encompass the range of training samples, while the red horizontal line in the plot indicates the smallest P Error of the theoretical classifier. The plot below illustrates how the MLP models' ability to more accurately approximating the class posteriors as the number of training samples rises results in a decline in error that is close to that achieved by the perfect classifier.



Appendix:

Question 1:
Data Generation:

```
import numpy as np
import pandas as pd
from numpy.linalg import inv
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def dataset_creation():
    Ntrain = [100, 200, 500, 1000, 2000, 5000]
    for Nsamples in Ntrain:
        data, labels = generateData(Nsamples)
#          stacking the data (3 X 100) and labels (1 X 100) to create a data 
set of size 100 X 4
        data_set = pd.DataFrame(np.transpose(np.vstack((data, labels))))
        plot3(data, labels, 'T')
#          stroing the egenrated data set in a .csv file
        filename = 'training_dataset' + str(Nsamples) + '.csv'
        data_set.to_csv(filename, index = False)

#      creating validation data sets
    NValidation = 100000
    data, labels = generateData(NValidation)
    plot3(data, labels, 'V')
    data_set = pd.DataFrame(np.transpose(np.vstack((data, labels))))
    filename = 'validation_dataset' + str(NValidation) +'.csv'
    data_set.to_csv(filename, index = False)

def generateData(N):
    gmmParameters = {}
#      given that priors are uniform so using 1/4 for each
    gmmParameters['priors'] = [0.25, 0.25, 0.25, 0.25] # priors should be a 
row vector
    gmmParameters['meanVectors'] = np.array([[0, 0, 0], [0, 0, 3], [0, 3, 0], 
[3, 0, 0]])
    gmmParameters['covMatrices'] = np.zeros((4, 3, 3))
    gmmParameters['covMatrices'][0,:,:] = np.array([[1, 0, -3], [0, 1, 0], [-
3, 0, 15]])
    gmmParameters['covMatrices'][1,:,:] = np.array([[8, 0, 0], [0, .5, 0], [0, 
0, .5]])
    gmmParameters['covMatrices'][2,:,:] = np.array([[1, 0, -3], [0, 1, 0], [-
3, 0, 15]])
    gmmParameters['covMatrices'][3,:,:] = np.array([[8, 0, 0], [0, 1, 0], [3, 
0, 15]])
    x, labels = generateDataFromGMM(N,gmmParameters)
```

```python
    return x, labels

def generateDataFromGMM(N,gmmParameters):
#    Generates N vector samples from the specified mixture of Gaussians
#    Returns samples and their component labels
#    Data dimensionality is determined by the size of mu/Sigma parameters
    priors = gmmParameters['priors'] # priors should be a row vector
    meanVectors = gmmParameters['meanVectors']
    covMatrices = gmmParameters['covMatrices']
    n = meanVectors.shape[1] # Data dimensionality
    C = len(priors) # Number of components
    x = np.zeros((n,N))
    labels = np.zeros((1,N))
    # Decide randomly which samples will come from each component
    u = np.random.random((1,N))
    thresholds = np.zeros((1,C+1))
    thresholds[:,0:C] = np.cumsum(priors)
    thresholds[:,C] = 1
    for l in range(C):
        indl = np.where(u <= float(thresholds[:,l]))
        Nl = len(indl[1])
        labels[indl] = (l)*1
        u[indl] = 1.1
        x[:,indl[1]] =
np.transpose(np.random.multivariate_normal(meanVectors[l,:],
covMatrices[l,:,:], Nl))
    labels = np.squeeze(labels)
    return x,labels

def plot3(data, labels, dtype):
#   from matplotlib import pyplot
#   import pylab
#   from mpl_toolkits.mplot3d import Axes3D
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.scatter(data[0,labels == 0], data[1,labels == 0], data[2,labels == 0],
marker='o', color='b')
    ax.scatter(data[0,labels == 1], data[1,labels == 1], data[2,labels == 1],
marker='^', color='r')
    ax.scatter(data[0,labels == 2], data[1,labels == 2], data[2,labels == 2],
marker='*', color='y')
    ax.scatter(data[0,labels == 3], data[1,labels == 3], data[2,labels == 3],
marker='+', color='g')
    ax.set_xlabel("x1")
    ax.set_ylabel("x2")
    ax.set_zlabel("x3")
    if (dtype == 'T'):
        ax.set_title('Training Dataset')
    else:
        ax.set_title('Validation Dataset')
    plt.show()

def predict(data, e):
    return np.matmul(e[:,0], pow(data,3)) + np.matmul(e[:,1], pow(data,2)) + 
np.matmul(e[:,2], pow(data,1)) + np.matmul(e[:,3], np.ones((2,1)))

dataset_creation()
```

MLP K-Fold Cross-Validation:

```
# Load the necessary Python libraries
# such as NumPy, Pandas, Matplotlib, Scikit-learn, and PyTorch.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import torch
import torch.nn as nn
from torch.utils.data import TensorDataset, DataLoader

# Define the MLP structure with two layers, a hidden layer, and an output 
layer.
# The number of perceptrons in the hidden layer should be determined by cross-
validation.
# You can experiment with different smooth-ramp activation functions such as
# ISRU, Smooth-ReLU, ELU, etc., to see which one works best for your problem.
class MLP(nn.Module):
    # def __init__(self, *args, **kwargs) -> None:
    #     super().__init__(*args, **kwargs)
    def __init__(self, input_dim, hidden_dim, output_dim) -> None:
        super().__init__()
        self.layer1 = nn.Linear(input_dim, hidden_dim)
        self.activationMethod = nn.ELU(alpha = 1.0)
        self.layer2 = nn.Linear(hidden_dim, output_dim)
        self.softmaxRegression = nn.Softmax(dim = 1)

    def forward(self, x):
        x = self.layer1(x.float())
        x = self.activationMethod(x)
        x = self.layer2(x)
        x = self.softmaxRegression(x)
        return x
```

```
# Train your MLP using the training set.
# During training, you will need to feed the input data forward
# through the MLP to compute the output, then backpropagate the
# error to update the weights. Repeat this process until the MLP converges,
# or until a predetermined stopping criterion is met.
def train(model, loss_fn, optimizer, train_loader):
    # Set the model to train mode
    model.train()
    # Iterate over the batches of training data
    for batch in train_loader:
        # Zero the gradients for this batch
        optimizer.zero_grad()
        # Compute the model's predictions for this batch
        inputs, labels = batch
        outputs = model(inputs.float())
        # Convert the labels tensor to Long type
        labels = labels.long()
        # Compute the loss between the predictions and the ground-truth labels
        loss = loss_fn(outputs, labels)
        # Backpropagate the loss and update the weights
        loss.backward()
        optimizer.step()

    return outputs, loss

# Evaluate the model's performance on the validation set after each epoch
def evaluate(model, loss_fn, test_loader):
    model.eval()
    with torch.no_grad():
        total_loss = 0
        total_correct = 0
        total_samples = 0
        for batch in test_loader:
            inputs, labels = batch
            outputs = model(inputs.float())
            # Convert the labels tensor to Long type
            labels = labels.long()
            loss = loss_fn(outputs, labels)
            total_loss += loss.item() * inputs.size(0)
            total_correct += (outputs.argmax(dim=1) == labels).sum().item()
            total_samples += inputs.size(0)
        val_loss = total_loss / total_samples
        val_acc = total_correct / total_samples
    return val_loss, val_acc, outputs

# Randomly initialize weights
def init_weights(m):
    if type(m) == nn.Linear:
```

```python
        nn.init.xavier_uniform_(m.weight)

def plotBarGraph(average_loss):
    # Plot a bar graph to decide optimal number of perceptrons
    plt.bar(np.linspace(1,10,10), average_loss)
    plt.xlabel("Number of Perceptrons")
    plt.ylabel("Average Loss after 10-Fold cross validation")
    plt.title("Plot of Error vs No of Perceptrons in the Hidden Layer")
    plt.show()

# Load your dataset into a Pandas dataframe or NumPy array.
data = pd.read_csv('training_dataset5000.csv')
input_features = data.iloc[:, : -1].to_numpy()
labels = data.iloc[:, -1].to_numpy()

#X_train, X_test, y_train, y_test = train_test_split(data.iloc[:, : -
1].to_numpy(), data.iloc[:, -1].to_numpy(), test_size=0.2, random_state=42)
batch_size = 32
num_folds = 10
kf = KFold(n_splits=num_folds)

minValidationLoss = float("inf") # Assining a large value to minValidationLoss
(+ infinity)
maxValidationAccuracy = float("-inf") # Assining a large value to
maxValidationAccuracy (- infinity)
optimal_no_of_Perceptrons = 0
average_loss = []

for perceptrons in range(1, 11):
    fold_loss = []
    fold_accuracy = []
    print('------------------------------------------------------------------
--')
    print(f"Number of Neurons / Perceptrons in the Hidden Layer :
{perceptrons} ")
    print('------------------------------------------------------------------
--')
    model = MLP(input_features.shape[1], perceptrons, len(np.unique(labels)))
    model.apply(init_weights)

    # Define the loss function, which in this case, is cross-entropy loss.
    # The loss function will be used to evaluate how well your MLP is
performing.
    import torch.nn.functional as F
    loss_fn = F.cross_entropy

    # Define the optimizer, which will be used to update the weights of your
MLP.
```

```
    # The most commonly used optimizer is stochastic gradient descent (SGD),
    # which updates the weights after each batch of training examples.
    import torch.optim as optim
    learning_rate = 0.001
    optimizer = optim.SGD(model.parameters(), lr=learning_rate, momentum =
0.9)

    for fold, (train_index, test_index) in
enumerate(kf.split(input_features)):
        # print(f'Fold {fold + 1}')
        # Split data into train and test sets for the current fold
        X_train, X_test = input_features[train_index],
input_features[test_index]
        y_train, y_test = labels[train_index], labels[test_index]

        # Create PyTorch data loaders for the train and test sets
        train_dataset = TensorDataset(torch.from_numpy(X_train),
torch.from_numpy(y_train))
        train_loader = DataLoader(train_dataset, batch_size=batch_size,
shuffle=True)

        test_dataset = TensorDataset(torch.from_numpy(X_test),
torch.from_numpy(y_test))
        test_loader = DataLoader(test_dataset, batch_size=batch_size,
shuffle=False)

        for epoch in range(50):
            # Forward pass
            outputs, loss = train(model, loss_fn, optimizer, train_loader)

            # Evaluate your MLP using the testing set.
            # This will give you an idea of how well your MLP generalizes to
new data.
            validationLoss, validationAccuracy, predictions = evaluate(model,
loss_fn, test_loader)

            # Print the epoch number, training loss, and validation loss and
accuracy
            # if epoch % 200 == 0:
        # print(f'Epoch {epoch}: TRAINING LOSS = {loss:.4f}, VALIDATION LOSS =
{validationLoss:.4f}, VALIDATION ACCURACY = {validationAccuracy:.4f}')

        # Evaluate the model's performance on the current fold and save the
error
        loss, accuracy, predictions  = evaluate(model, loss_fn, test_loader)
        fold_loss.append(loss)
        fold_accuracy.append(accuracy)
```

```
     # Calculate the mean error across all folds for the current number of 
perceptrons
    mean_error = np.mean(fold_loss)
    mean_accuracy = np.mean(fold_accuracy)
    print(f'Mean error for {perceptrons} perceptrons: {mean_error:.4f}')
    average_loss.append(mean_error)

    if (minValidationLoss > mean_error):
        minValidationLoss = min(mean_error, minValidationLoss)
        maxValidationAccuracy = max(mean_accuracy, maxValidationAccuracy)
        optimal_no_of_Perceptrons = perceptrons

print(f'Optimal number of Perceptrons = {optimal_no_of_Perceptrons} : 
VALIDATION LOSS = {minValidationLoss:.4f}, VALIDATION ACCURACY = 
{maxValidationAccuracy:.4f}')

# Finally, visualize the results using Matplotlib or other visualization tools
# to better understand the performance of your MLP.
plotBarGraph(average_loss)

# Experiment with different parameters,
# such as the number of perceptrons in the hidden layer,
# the learning rate of the optimizer, the number of training epochs, etc.,
# to see how they affect the performance of your MLP.

# Load your dataset into a Pandas dataframe or NumPy array.
data = pd.read_csv('training_dataset500.csv')
X_train = data.iloc[:, :-1].to_numpy()
y_train = data.iloc[:, -1].to_numpy()

# Load your dataset into a Pandas dataframe or NumPy array.
data = pd.read_csv('validation_dataset100000.csv')
X_test = data.iloc[:, :-1].to_numpy()
y_test = data.iloc[:, -1].to_numpy()

model = MLP(input_features.shape[1], optimal_no_of_Perceptrons, 
len(np.unique(labels)))
model.apply(init_weights)
loss_fn = F.cross_entropy
learning_rate = 0.001
optimizer = optim.SGD(model.parameters(), lr=learning_rate,  momentum = 0.9)

# Create PyTorch data loaders for the train and test sets
train_dataset = TensorDataset(torch.from_numpy(X_train), 
torch.from_numpy(y_train))
train_loader = DataLoader(train_dataset, shuffle=True)
batchSize = X_test.shape[0]
```

```
test_dataset = TensorDataset(torch.from_numpy(X_test),
torch.from_numpy(y_test))
test_loader = DataLoader(test_dataset, batch_size = batchSize, shuffle=False)

for epoch in range(250):
    print(epoch)
    # Forward pass
    outputs, loss = train(model, loss_fn, optimizer, train_loader)

    # Evaluate your MLP using the testing set.
    # This will give you an idea of how well your MLP generalizes to new data.
    #validationLoss, validationAccuracy, predictions = evaluate(model,
loss_fn, test_loader)

y_pred = torch.zeros(0, dtype=torch.long, device='cpu')
# Evaluate the model's performance on the current fold and save the error
loss, accuracy, predictions = evaluate(model, loss_fn, test_loader)
print(f"FINAL LOSS IS {loss}, ACCURACY IS {accuracy}")
y_pred = torch.cat([y_pred, predictions.argmax(1)])
ConfusionMatrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(ConfusionMatrix)
error = (1 - (ConfusionMatrix[0][0] + ConfusionMatrix[1][1] +
ConfusionMatrix[2][2] + ConfusionMatrix[3][3]) / X_test.shape[0])
print("Error:")
print(error)
```

Theoretical Classifier:

```
import random
import numpy as np
import pandas as pd
import numpy.matlib
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
from mpl_toolkits.mplot3d import Axes3D
np.set_printoptions(threshold=np.inf)

priors = np.array([[0.25, 0.25, 0.25, 0.25]]) # priors should be a row vector
meanVectors = np.array([[0, 0, 0], [0, 0, 3], [0, 3, 0], [3, 0, 0]])
covMatrices = np.zeros((4, 3, 3))
covMatrices[0,:,:] = np.array([[1, 0, -3], [0, 1, 0], [-3, 0, 15]])
covMatrices[1,:,:] = np.array([[8, 0, 0], [0, .5, 0], [0, 0, .5]])
covMatrices[2,:,:] = np.array([[1, 0, -3], [0, 1, 0], [-3, 0, 15]])
covMatrices[3,:,:] = np.array([[8, 0, 0], [0, 1, 0], [3, 0, 15]])

# compute class condtional pdf
def compute_class_conditional_pdf(labels, no_labels, no_samples):
    P_x_given_L = np.zeros(shape = [no_labels, no_samples])
    unq_ls = len(np.unique(labels))
    for i in range(unq_ls):
        P_x_given_L[i, :] = multivariate_normal.pdf(input_features,
meanVectors[i, :], covMatrices[i, :, :])
    return P_x_given_L

# compute Confusion Matrix
def compute_confusion_matrix (No_labels, class_labels):
    cm = np.zeros(shape = [No_labels, No_labels])
    for i in range(No_labels):
        for j in range(No_labels):
            if j in class_labels and i in class_labels:
                cm[i, j] = (np.size(np.where((i == Decision) & (j ==
class_labels)))) / np.size(np.where(class_labels == j))
    return cm

# Load your dataset into a Pandas dataframe or NumPy array.
data = pd.read_csv('validation_dataset100000.csv')
# Input features are x and class labels are y
input_features = data.iloc[:, :-1].values
labels = data.iloc[:, -1].values
no_labels = len(np.unique(labels))
no_samples = input_features.shape[0]

P_x_given_L = compute_class_conditional_pdf(labels, no_labels, no_samples)
# Compute Class Posteriors using priors and class conditional PDF
P_x = np.matmul(priors, P_x_given_L)
class_posteriors = (P_x_given_L * (np.matlib.repmat(np.transpose(priors), 1,
no_samples))) / np.matlib.repmat(P_x, no_labels, 1)

# Define 0-1 Loss Matrix
loss_matrix = np.ones(shape = [no_labels, no_labels]) - np.eye(no_labels)

# Evaluate Expected risk and decisions based on minimum risk
expected_risk = np.matmul(loss_matrix, class_posteriors)
Decision = np.argmin(expected_risk, axis = 0)
avg_exp_risk = np.sum(np.min(expected_risk, axis = 0)) / no_samples
print(f'Average Expected: {avg_exp_risk}')
confusion_matrix = compute_confusion_matrix (no_labels, labels)
print("Confusion Matrix:")
print(confusion_matrix)
```

Code for Plotting:

```
import numpy as np
import matplotlib.pyplot as plt
# x represents number of training samples
```

```python
x = [100, 200, 500, 1000, 2000, 5000]
# elements in error correspond to loss produced by ML model with training
samples in x
error = [25.79, 25.49, 23.36, 23.17, 21.71, 25.09]
# Plot graph to compare theoretical classifier and various MLP models
plt.semilogx(x, error, marker = "*", markersize = 18)
plt.axhline(y = 17.903, color = 'b', linestyle = '-')
plt.xlabel("Number of Samples (Semilog axis)")
plt.ylabel("Emperically Estimated P Error")
plt.title("Plot to Compare Errors of the trained models")
plt.show()
```